Aufgabe 3: Voll daneben

Team: Paddy Jo
Team-ID: 00920

26. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Als erstes haben wir uns mit dem Problem an sich befasst und überlegt, ob Al überhaupt Verlust macht. Dazu haben wir zuerst überlegt, wie es bei 1000 Teilnehmern aussehen würde, die alle auf eine andere Zahl setzen. Dann sind Al's Einnahmen $1000 * 25\$ = 25000\$$. Bei 1000 verschiedenen gesetzten Zahlen (im Folgenden als Stellen oder Elemente bezeichnet) kann man die kleinstmögliche Auszahlung erreichen, indem man immer gleich viele Stellen zu einer Gruppe (im Folgenden als Sublist bezeichnet) zusammenfasst. Dadurch hat man 10 Sublists mit 100 Stellen pro Sublist. Setzt Al nun seine 10 Glückszahlen in die Mitte jeder Sublist, also auf 50, 150, 250, etc., beträgt Al's Gewinn 1275$. Er macht also selbst in einem schlechten Fall noch Gewinn. Gehen wir nun davon aus, dass bei einer ungleichen Verteilung zwar in einer Sublist höhere Kosten entstehen, dafür in einer anderen geringere, ist dies der ein sehr guter Lösungsweg. Zur Lösung des Problems haben wir also nun den Ansatz verfolgt, dass es generell effizient ist, die Stellen in 10 Sublisten zusammenzufassen. Damit dies sinnvoll ist, muss die Liste der Stellen zuerst sortiert werden. Ist dies geschehen, bildet man die Gruppen. In jeder Gruppe sind nun die Stellen, die möglichst nah aneinander liegen. Da es jedoch vorkommen kann, dass 2 Stellen sehr nah beieinander liegen aber trotzdem in zwei verschiedenen Sublists liegen, muss danach noch überprüft werden, ob es sinnvoll ist, einzelne Stellen noch unter den Sublisten zu verschieben.

2 Umsetzung

Wie bereits eingehend erwähnt, müssen die Stellen zuerst sortiert werden. Am sinnvollsten ist es, sie aufsteigend zu sortieren. Danach wird berechnet, wie viele Stellen in eine Sublist müssen. Dazu wird die Anzahl der Stellen durch 10 geteilt. Falls keine ganze Zahl dabei herauskommt, wird diese abgerundet. Dadurch kann es passieren, dass eine elfte Sublist entsteht, in der die übrigen Elemente liegen. Da Al jedoch nur 10 Glückszahlen hat, darf es auch nur 10 Sublists geben. Daher wird in diesem Fall errechnet, bei welchen Sublists es günstig wäre, ein weiteres Element hinzuzufügen. Dies lösen wir, indem wir schauen, für welche der 10 ersten Sublists die jeweils nächste Stelle den kleinsten Abstand hat. Haben wir eine Sublist gefunden, für die der Abstand im Vergleich zu allen anderen am kleinsten ist, fügen wir die Stelle zu dieser Sublist hinzu. Da sich dadurch jedoch alle nachfolgenden Sublists um eine Stelle nach hinten verschieben, können wir nicht alle Elemente aus der elften Sublists auf einmal einfügen, sondern müssen zwischen jedem Vorgang alle Sublisten und auch die Abstände neu berechnen, um eine in dieser Hinsicht optimale Verteilung der übrigen Elemente zu erreichen. Die Verteilung ist nun schon relativ gut. Um sie jedoch noch zu verbessern, wird der Mittelwert der Abstände zwischen den Stellen jeder Sublist

berechnet. Ist der Abstand zwischen der letzten Stelle einer Sublist und der ersten Stelle der nächsten Sublist kleiner als der Mittelwert beider Sublists und der Mittelwert der ersten Sublist kleiner als der der zweiten, so wird die erste Stelle aus der zweiten Sublist zur ersten hinzugefügt. Die anderen Sublists verschieben sich dadurch nicht, in der zweiten Sublist ist so lediglich ein Element weniger und in der ersten eben ein Element mehr. Dieser Vorgang wird für alle 10 Sublists durchgeführt. Danach sollten die Stellen optimal auf die Sublists verteilt sein. Nun müssen nur noch die Glückszahlen in die Mitte jeder Sublist gesetzt werden. Falls in einer Sublist eine ungerade Anzahl an Stellen ist, wird abgerundet. Dies hat im durchschnitt keinen Einfluss auf das Endergebnis.

3 Beispiele

Das erste Beispiel der BwInf-Website führt zu folgenden Glückszahlen:
45, 140, 235, 330, 425, 520, 615, 710, 805, 900, 970
Dadurch erzielt Al einen Gewinn von 500$.
Für das zweite Beispiel werden die folgenden Glückszahlen errechnet:
42, 123, 226, 336, 393, 505, 584, 754, 857, 926
Der Gewinn beläuft sich dabei auf 299$.
Für das dritte Beispiel sind folgende Glückszahlen optimal:
80, 220, 280, 380, 480, 540, 640, 720, 800, 900
Der Gewinn beläuft sich auf 300$.

Bei unter 10 Teilnehmern hat Al logischerweise keinen Verlust, da er auf jede Stelle eine Glückszahl setzen kann.
Bei mehr als 10 Teilnehmern ist es für Al natürlich günstig, wenn mehrere Teilnehmer auf die gleiche Stelle setzen.
Passiert dies nicht, ist eine Haufenbildung auch günstig für Al, da er seine Glückszahl jeweils in die Mitte eines Haufens setzen kann.
Im schlimmsten Fall für Al sind entweder 20 Haufen gleichmäßig verteilt oder einzelne Zahlen gleichmäßig verteilt, was zu einem ähnlichen Ergebnis führt. Ist z.B. jede zehnte Stelle belegt, führt dies für Al zu einem Gewinn von nur 150$. Jedoch macht er selbst in diesem Worst-Case noch Gewinn.

4 Quellcode

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Paddy Jo
@title BwInf 2018 A3
"""
import math #Für das Runden u.a. der Mittelwerte
import sys #Für die Übergabe von Kommandozeilenargumenten
#Der folgende Abschnitt dient zum Einlesen der Beispiele
readLuckyList = []

if len(sys.argv)!=2:
    print("Benutzung: Aufgabe3.py <pfad\\zur\\datei.txt>")
    sys.exit(0)

with open(sys.argv[1]) as file:
    readLuckyList = file.readlines()

luckyList = [] #Die luckyList ist die Liste mit den Zahlen, auf die die Besucher gesetzt haben.
luckyNumbers = [] #Die luckyNumbers sind die Zahlen, auf die Al setzen sollte.
for i in readLuckyList:
    i=i[:-1]
    luckyList.append(int(i))
luckyList.sort()

#Wenn die Liste weniger als 10 Einträge hat,
#werden die Glückszahlen direkt auf die gesetzten Stellen gesetzt:
if len(luckyList) <= 10:
    for i in luckyList:
        luckyNumbers.append(i)
```

```
    for y in range(10-len(luckyNumbers)):
        luckyNumbers.append(y+1)
else:
    """
    Der folgende Abschnitt dient dazu, eine bestimmte Anzahl
    von gesetzten Stellen zu einer Liste zusammenzufassen
    (Im folgenden werden diese Listen als "Sublist" bezeichnet).
    Danach werden die 10 Sublisten in die intervallList eingefügt,
    sodass man über die intervallList am Ende alle
    Sublists abrufen kann.
    """

    minVal = math.floor(len(luckyList)/10)  #minVal ist die Menge der Zahlen,
                                            #die mindestens in eine Sublist muss,
                                            #sodass man am Ende alle Stellen in 10
                                            #Sublisten aufgeteilt hat.

    intervallList = []
    tempIntervallList = []
    for i in luckyList:
        tempIntervallList.append(i)
        if len(tempIntervallList) == minVal:
            intervallList.append(tempIntervallList)
            tempIntervallList = []
    intervallList.append(tempIntervallList)

    if len(intervallList) == 10:
        while intervallList[10] is not None:    #Wenn Elemente übrig waren,
                                                #welche nun in einer 11. Sublist sind:
            """
            Im folgenden Abschnitt werden die "übrigen" Elemente aus der 11. Sublist
            möglichst effizient auf die ersten 10 Sublisten verteilt.
            Als erstes wird berechnet, wie die Abstände der Randelemente der
            ersten 10 Sublisten sind.
            """
            v2List = []
            for i in range(1, len(intervallList)-1):
                v2List.append(intervallList[i][0] - intervallList[i-1][len(intervallList[i-1])-1])
            """
            Nun wird geschaut, bei welchen Sublisten diese Abstände am kleinsten sind.
            Der erste Wert in der v2List ist dabei der Abstand zwischen
            dem letzten Element in der ersten Sublist und dem ersten Element in der zweiten Sublist.
            Nach jeder Zuteilung eines weiteren Elements zu einer der Sublisten  wird der ganze
            Vorgang wiederholt, damit sich durch die generelle
            Verschiebung aller Sublisten keine Fehler ergeben.
            """
            for repeat in range(len(intervallList[10])):
                tempMinVal = v2List[0]
                tempPosition = 0
                for y in range(len(v2List)-1):
                    if v2List[y] > tempMinVal and v2List[y] != -1:
                        tempMinVal = v2List[y]
                        tempPosition = y
            #Hier werden jetzt die Sublisten neu errechnet, wobei die Subliste an der Stelle
            #tempPosition ein Element mehr zugeteilt bekommt:
            intervallList = []
            tempIntervallList = []
            i = 0
            while i < len(luckyList):
                tempIntervallList.append(luckyList[i])
                if len(tempIntervallList) == minVal:
                    if len(intervallList) == tempPosition:
                        tempIntervallList.append(luckyList[i+1])
                        i += 1
                    intervallList.append(tempIntervallList)
                    tempIntervallList = []
                i+=1
            intervallList.append(tempIntervallList)



    """
    Im folgenden Abschnitt werden die nun errechneten Sublists noch weiter optimiert,
```

```
    indem zuerst die mittlere Abweichung der Elemente in
    jeder Sublist berechnet wird. Danach wird die Abweichung zwischen dem letzten Element
    einer Liste und dem ersten Element der nächsten Liste
    errechnet. Ist diese Abweichung kleiner als die mittlere Abweichung beider Listen
    und die mittlere Abweichung der ersten Liste kleiner als
    die der zweiten, so wird das Element aus der zweiten Liste in die erste geschoben.
    """
    vList = []
    for sublist in range(10):
        tempVList = []
        for position in range(1, len(intervallList[sublist])-1):
            tempVList.append(intervallList[sublist][position] - intervallList[sublist][position-1])
        sum = 0
        for i in tempVList:
            sum += i
        mVal = sum/len(tempVList)
        vList.append(mVal)

    v2List = []
    for i in range(1, len(intervallList)-1):
        v2List.append(intervallList[i][0] - intervallList[i-1][len(intervallList[i-1])-1])

    for index in range(1, len(intervallList)-1):
        if vList[index] > vList[index-1]:
            if v2List[index-1] < vList[index] and v2List[index-1] < vList[index-1]:
                toBeMoved = intervallList[index][0]
                intervallList[index-1].append(toBeMoved)
                intervallList[index].remove(toBeMoved)



    #Jetzt wird immer der Wert in der Mitte jeder Sublist zur luckyNumbers Liste hinzugefügt.
    for i in intervallList:
        middle = len(i)/2
        middle = math.floor(middle)
        if middle != 0:
            luckyNumbers.append(i[int(middle)-1])

#Diese Funktion dient dazu, den Betrag einer Zahl zurückzugeben.
def pos(val):
    if val<0:
        val*=-1
    return val

#Mit dieser Funktion wird der Gewinn von Al berechnet. Dazu werden zuerst seine Einnahmen
#und dann seine Ausgaben berechnet. Die Differenz ist der Gewinn.
def calcWin():
    income = len(luckyList)*25
    loss= 0
    for i in luckyList:
        for y in luckyNumbers:
            if y>=i:
                diff1=pos(y-i)
                diff2=pos(luckyNumbers[luckyNumbers.index(y)-1]-i)
                loss+=min(diff1,diff2)
                break
    win=income-loss
    return win



print("Auf diese Nummern sollte Al setzen:",luckyNumbers)
print("damit sein Gewinn dieser ist:",calcWin(),"$")
```

Aufgabe3.py